

**REMARKS**

The Office Action dated March 14, 2005, has been reviewed in detail along with all references made of record. Reconsideration of the claims of the instant application is respectfully requested in view of the following remarks.

Applicant would like to extend its appreciation to the Examiner for the time and attention accorded this case. As will be set forth in detail herebelow, the issues raised by the Office in the outstanding Office Action, when reconsidered in light of the foregoing amendments and the following comments, should be resolved in Applicant's favor.

As of the mailing date of the outstanding Office Action, Claims 1-28 were pending in the instant application. By this Amendment, Claims 1, 2, 4, 7, 8, 14-17, 19-21, 24, 26 and 27 are amended.

Claims 1-7, 10, 11, 14-17, 20-24, 26 and 27 stand rejected under 35 U.S.C. 102(b) as "being anticipated by Marmelstein (U.S. Patent No. 5,187,788)..."

Independent Claim 1 broadly relates to a method for generating machine control instructions. A user input is read in to select an object from a library of objects, wherein the objects include one or more sets of machine control instructions. The selected object is connected to a network of objects which includes objects previously selected and connected by the user. Inputs and outputs of the selected object are identified, and such inputs and outputs are connected to inputs and outputs of other objects in the network. What results is the formation of an aggregate comprising the network of objects and the connections between connected objects. Machine control instructions are generated via employing instructions now contained in the network of objects. Responsive to this, updates are effected within the aggregate of the network of objects and the

connections between connected objects to accurately reflect any changes made to the machine control instructions generated in the generating step.

Some attendant advantages are apparent from this. Essentially, it may be stated that there is broadly contemplated, in accordance with at least one embodiment of the present invention, a software development system that essentially generates machine code instructions in a flexible and universal manner to substantially reduce the time and resources required to produce production-quality software. Preferably, one or more objects are used that contain machine control instructions. These objects can be freely modified and connected to one another without limiting further code generation.

A primary consequent advantage is that essentially any desired part of the system may be tested and debugged incrementally, without requiring the system to be completely coded. Unlike conventional arrangements, this feature is not limited to specific parts of the system, and can be applied to any desired portion of the system. Furthermore, these changes will be automatically reflected in any other representation with which the user chooses to view or edit the system. This allows incremental development to occur at every step of the development process, unlike conventional arrangements which support incremental development only for compilation of computer source code. This also allows parts of the system to be tested, edited, and debugged on remote machines over a network by several users simultaneously. The changes made are reflected immediately in each user's chosen representation.

It is respectfully submitted that Marmelstein does not at all teach or suggest the features recited by Claim 1. Generally, Marmelstein relates to graphic code generation in the context of an avionics program. Generally, graphic representations in the form of a flowchart are made available to a user for varying purposes. For instance, as will be further appreciated herebelow, such a graphic representation may initially provide an easily accessible and "digestible" overview of

programming code that has been developed by the user while the user can be availed of the opportunity to manipulate or interact with the graphic representation to amend this representation before code is actually generated.

As has been discussed in previous correspondence, any "updating" conducted in the context of Marmelstein can only be said to take place within a common database, or a collection of objects organized in such a way that a computer program can quickly select the desired pieces of data, or objects. For instance, during a process of selecting an APEX ("Avionics Program Export") object, the position of a mouse cursor on a screen determines what object is selected. Once the position of the mouse cursor is known, a search of the common database is performed for the appropriate APEX object whose position matches the mouse cursor's position (see column 10, lines 12-18).

Completely lacking in Marmelstein is the degree of versatility and operability that is enjoyed in connection with embodiments of the present invention as broadly defined by Claim 1. For this, it is important to recognize a crucial difference between the respective "spaces" in which updating occurs. As broadly defined in Claim 1, an aggregate comprising a network of objects and connections between objects represents the "space" in which updating may occur, not merely the much more restricted "space" of Marmelstein's common database. As it stands, Marmelstein's common database, at best, is reminiscent of the "library of objects" recited by Claim 1, that is, an original space from which objects are originally selected for subsequent inclusion in a broader network. As such, the aggregate of a network of objects and connections between objects, as recited in Claim 1, truly has no analog in Marmelstein. Accordingly, it is respectfully submitted that Marmelstein cannot possibly be construed as teaching or suggesting at step of updating within an aggregate of a network of objects and connections between connected objects.

Further comments also appear to be warranted with regard to the concept of "generating machine control instructions", which has also been discussed at length in previous correspondence.

Again, there is a crucial difference here between the embodiments of the present invention, as broadly defined by Claim 1, and Marmelstein. Marmelstein, again, works within a very limited framework for "generating" machine control instructions, let alone for how these are then used.

Along these lines, it is important to appreciate that, in accordance with embodiments of the present invention as broadly defined by Claim 1, machine control instructions are generated via employing instructions contained in the network of objects. Responsive to this, updates are effected within the aggregate of the network of objects and the connections between connected objects, to accurately reflect any changes made to the machine control instructions generated in the generating step.

Marmelstein, in stark contrast, does not appear to provide for updating the network of objects based on changes to generated machine control instructions. Indeed, if a user modifies the source code or binary code "generated" by the APEX system, then Marmelstein's high level graphical representation is not automatically updated. Similarly, other graphical program generators will generate source code from graphical interface elements, but modifying "generated" source code does not automatically modify these graphical interface elements. For example, adding a new function or object in the Ada source code generated by APEX does not add a new element and its corresponding connections in the graphical representation. Marmelstein thus may appear to teach the updating of a "network" of objects to reflect changes made to edited objects, but not changes made to code generated after the code is generated.

To explore this topic further, the simple use of a computer or the mere execution of functions in software programs does not "generate" any machine control instructions. In stark contrast, "generated" machine control instructions, as can be understood in the context of Claim 1, are generated at the very least from a network of objects, whereas the mere performance of a

function on a computer or in a software program represents the utilization of pre-existing machine control instructions rather than their generation.

Generally, the updating of objects and connections in Marmelstein only occurs when editing the high level network of objects in the graphical editor. In contrast, Claim 1 broadly embraces the automatic updating of a network of objects, and the connections in the network, to accurately reflect any changes made to the generated machine control instructions. If this feature were to be directly applied to the environment of Marmelstein, then one would see the generation of machine control instructions from objects such as the Ada source code generated by the "Generate Ada" command in APEX (in Marmelstein), and objects and connections making up the network would be updated when such generated machine control instructions are edited. However, the event handler contemplated by Marmelstein does not update such objects and connections that make up the network when these generated machine control instructions are edited but instead only allows the user to do so before generating the machine control instructions.

For its part, the mouse click described in Marmelstein does not in fact generate machine control instructions, but simply causes the operating system to call a function (which happens to contain pre-existing machine control instructions, not generated ones) to obtain the pointer and find the object in the database. Any arguably bona fide "generation of machine control instructions" does not occur until the "Generate Ada" command is selected by the user to produce Ada source code and that source code is compiled to binary code.

In view of the foregoing, it will be readily apparent that the methods and arrangements contemplated by Marmelstein are very far afield from the features recited by Claim 1. Accordingly, it is respectfully submitted that Claim 1 fully distinguishes over Marmelstein.

Independent Claim 21 broadly relates to a method for constructing a high-level object model from generated machine control instructions. A sequence of machine control instructions is read in.

A library of objects is searched for one or more matching objects configured for generating the sequence of machine control instructions already read in. Each sequence of machine control instructions so matched is parsed to determine the objects connected to the inputs and outputs of each matching object found in the library of objects. Each matching object found in the library of objects is connected to the other objects in a high-level model found in reading, searching and parsing.

Again, it is respectfully submitted that Marmelstein does not teach or suggest the above features. As such, the Office indicates that Marmelstein teaches "a method for constructing a high-level object model from generated machine control instructions, the method comprising the steps of: reading in a sequence of machine control instructions for performing one or more functions". This, however, does not characterize Marmelstein.

As cited by the Office, Marmelstein discloses the selecting of objects from the database list at the start of operation (Figure 13 and Column 10, lines 26-32). This has no relation to reading in a sequence of machine control instructions, as recited in Claim 21. It is inherent that for a sequence of machine control instructions to be "read in", then such a sequence is by definition preexisting. An upshot is that a capability is provided for taking preexisting, generated machine control instructions and essentially "decomposing" them into a network of objects that can be edited. (For an illustrative, non-restrictive example of this phenomenon, see the instant specification at p. 13, second full paragraph). Accordingly, the reading in of a sequence of machine control instructions effectively provides a model or template that can be modified or have new code generated from it, by searching an objects database for objects matching those of the model sequence originally read in.

In stark contrast, while Marmelstein does select objects with which to generate code, it does not first read in a model or template code to match with later selected objects. Marmelstein, for its

part, does not read in a sequence of generated machine control instructions to construct a high level object model that corresponds to those instructions. Marmelstein only reads in an object indicated by the user, yet does not automatically select or create such an object by reading in machine control instructions that would be generated by such an object. Essentially, Claim 21 broadly embraces the use of generated machine control instructions that are the result of a previous generation process wherein there are automatically identified – and connected - a corresponding network of objects that could then be edited and used to generate a new sequence of machine control instructions.

In view of the foregoing, it is respectfully submitted that Claim 21 fully distinguishes over Marmelstein.

By virtue of dependence from Claims 1 and 21, it is respectfully submitted that Claims 2-7, 10, 11, 14-17, 20, 22-24, 26 and 27 fully distinguish over Marmelstein as do Claims 1 and 21. However, some further comments on the dependent claims may be warranted.

Regarding Claim 2, Marmelstein's "generate Ada" command in APEX may produce source code for a completed representation of the system, but it does not also modify the system to reflect changes that may have been made to parts of the generated source code, or to generated binary code (in the case that the system is used to generate binary code). Once the "generate Ada" command is executed, the network of objects used to generate the Ada source code must be manually edited and all of the source code re-generated using the entire network if the user wishes to automatically generate a modified version of the source code.

Marmelstein does not teach or suggest any method for automatically incorporating into the graphical representation of the code any changes made to the generated source code. In addition, Marmelstein only teaches a method for generating the entire network of objects for a complete subprogram, while there is broadly contemplated, in accordance with at least one presently preferred embodiment of the present invention as defined by Claim 2, the generation and updating

of individual objects in the network of objects. This can therefore produce partial and incomplete functions that can later be combined with other partial functions generated from other objects as desired.

As to Claim 4, Marmelstein can be said to load code libraries during "generation", while at least one presently preferred embodiment of the present invention as broadly defined by Claim 4 essentially involves the generation of code to load libraries at runtime when the generated code is actually executed. Essentially, Claim 4 relates to generating code that contains machine control instructions that load the libraries (which conceivably could include both code and auxiliary data) that are represented by objects. The loading of the libraries essentially occurs when the generated machine code is executed, not when it is generated, unlike in Marmelstein where such loading occurs only during generation of machine code instructions.

Regarding Claim 6, Marmelstein does not teach the generated machine control instructions as computer instructions belonging to an instruction set architecture. Particularly, Marmelstein's APEX program generates Ada source code, and yet Ada is not an instruction set architecture but is a programming language. While the Ada source code generated by APEX could be compiled into binary computer instructions belonging to an instruction set architecture (such as the Sun SPARC instruction set), this compiling method would not allow direct correspondence between an object in a network of objects and the compiled binary computer instructions, since Ada does not support partial compilation of individual lines of an Ada program. A given object in the network of objects will not directly correspond to a block of compiled binary computer instructions, since an Ada compiler will not generate separate binary computer instructions for each object. Such a compiler may even optimize out a portion or all of the source code generated by some objects. By contrast, Claim 6 is essentially directed to computer instructions belonging to an instruction set architecture being directly generated from individual objects in the network of objects.



As to Claim 14, Marmelstein only teaches using the positioning of virtual objects (icons) on a computer display corresponding to objects in the library. By contrast, Claim 14 essentially relates to using physical objects and their relative placement in the "real world", wherein each physical object corresponds directly to a specific object in the library, as a result of which the physical relationships between tangible, physical objects in the "real world" cause the corresponding connections to be made in the network of objects on a computer system.

As to Claims 17 and 27, Marmelstein only visually indicates which objects are currently being used to generate code, while Claims 17 and 27 visually indicate the actual execution of code at runtime, not during generation. Essentially, Claims 17 and 27 relate to the ability to monitor the actual execution of a program's code that corresponds to each object in the system, which would visually indicate the flow of data and program execution as the binary code is executed by indicating which "active objects" correspond to the data and binary code currently being used. Unlike in Marmelstein, the monitoring occurs during execution of the generated code, rather than during generation of the code.

In view of the foregoing, it is respectfully submitted that Claims 1-7, 10, 11, 14-17, 20-24, 26 and 27 fully distinguish over the applied art. Accordingly, reconsideration and withdrawal of the present rejection are hereby respectfully requested.

Claims 8 and 25 stand rejected under 35 U.S.C. 103(a) in view of Marmelstein and Brender et al. Claims 12 and 13 stand rejected under 35 U.S.C. 103(a) in view of Marmelstein and Lithicum et al. Finally, Claims 9, 18, 19 and 28 stand rejected under 35 U.S.C. 103(a) in view of Marmelstein and Zink et al.

It is respectfully submitted that Claims 8, 9, 12, 13, 18, 19, 25 and 28 are allowable in view of the apparent allowability of independent Claims 1 and 21. At the same time, it is submitted that the secondary references, for their part, do not strengthen the present rejections in any event.

Brender teaches a cross-domain call jacketing system while Lithicum teaches a haptic interface for detecting and avoiding collisions between objects in a virtual space. Zink, on the other hand, teaches a graphical development system for software or software/hardware hybrids with intelligent development components. As such, none of these secondary references, in any reasonable combination with Marmelstein, serves to anticipate or render obvious the claims at issue in the present rejections.

In view of the foregoing, it is respectfully submitted that Claims 8, 9, 12, 13 18, 19, 25 and 28 fully distinguish over the applied art. Accordingly, reconsideration and withdrawal of the present rejection are hereby respectfully requested.

It will be noted, in closing, that several amendments have been made to the claims, beyond those discussed herein, to impart even greater clarity to the claims and/or to render some minor linguistic and/or stylistic changes.

**References Made of Record but not Applied:**

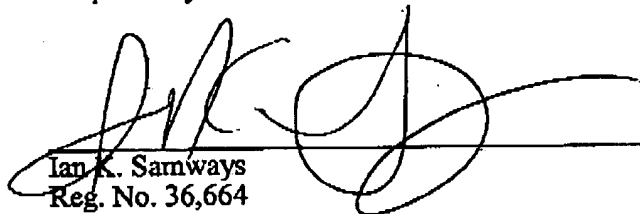
The references made of record and considered by the Office to be "pertinent to applicant's disclosure" have been reviewed. Applicant acknowledges that the Office has deemed such references not sufficiently relevant to have been relied upon in the outstanding Office Action. However, to the extent that the Office may apply such references against the claims in the future, Applicant is prepared to fully respond thereto.

\* \* \*

In summary, Applicant respectfully submits that the instant application, including Claims 1-28, is presently in condition for allowance. Notice to the effect is hereby earnestly solicited.

Respectfully submitted,

Dated: June 13, 2005



Ian K. Samways  
Reg. No. 36,664

REED SMITH LLP  
P.O. Box 488  
Pittsburgh, PA 15230  
(412) 288-4160

Agent for Applicant